Article history:
Received 29 July 2024
Revised 25 September 2024
Accepted 04 October 2024
Published online 24 October 2024

Journal of Resource Management and Decision Engineering

Volume 3, Issue 4, pp 116-135

A Machine Learning-Based Approach for Link Recovery in Smart Grids Using Software-Defined Networking

Nasser Mozayani [1*], Hemila Vali [2]

[1] Associate Professor, Department of Computer Science, Faculty of Computer Engineering, Iran University of Science and Technology, Tehran, Iran
[2] Faculty, Department of Computer, School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran
*** Corresponding author email address**: Mozayani@iust.ac.ir

A r t i c l e I n f o

Article type:
Original Research

How to cite this article:
Mozayani, N., & Vali, H. (2024). A Machine Learning-Based Approach for Link Recovery in Smart Grids Using Software-Defined Networking. *Journal of Resource Management and Decision Engineering, 3*(4), 116-135.
https://doi.org/10.61838/kman.jrmde.3.4.13

A B S T R A C T

Network failures can cause serious damage and disrupt communications, which in turn significantly reduces service reliability. Among the various causes of network failure, link failure between network components is one of the most common. For intelligent, autonomous operation without human intervention, a smart grid must have a flexible communication infrastructure capable of rapidly detecting and repairing link failures. A software-defined networking (SDN) approach offers this capability. SDN enables centralized control and decouples the data and control planes, facilitating dynamic management and rapid response to failures. This technology enhances the speed of rerouting and path recovery during link failures by providing centralized and optimized decision-making. Central controllers in SDN architectures can select new paths for data flows in the event of failure, thereby reducing recovery time and packet loss rates. In this study, we design a module for the SDN controller equipped with link failure detection features and implement automatic recovery using the Q-learning algorithm. In smart grid environments, packet loss rates are critical because any data loss can degrade service quality and hinder the reception of essential information required for system control in crisis situations, ultimately reducing the grid's ability to respond to instabilities. The core focus of this research is reducing the packet loss rate, as data loss severely impacts network stability and efficiency. When comparing the proposed method to a reference study, the packet loss rate, recovery time, and algorithm execution time in the German topology decreased by 66.62%, 85.99%, and 91.99%, respectively. Similarly, in the U.S. topology, these metrics were reduced by 90.50%, 76.99%, and 98.99%, respectively. Additionally, compared to the normal state, the packet loss rate was reduced by an average of 67.95% in the German topology and 13.15% in the U.S. topology.

Keywords: *Smart grid, software-defined networking (SDN), Q-learning algorithm, fault management.*

1. Introduction

T

he rapid evolution of electrical infrastructure into highly automated and intelligent systems has made the concept of the smart grid increasingly vital to modern energy management and distribution. Smart grids are complex cyber-physical systems that incorporate bi-directional flows of information and electricity to enable adaptive, efficient, and sustainable power distribution, especially under dynamic and uncertain conditions (Shafiullah et al., 2013). However, this increasing sophistication also brings about critical vulnerabilities—especially concerning communication reliability and routing resilience, particularly in the face of link failures. In response to these challenges, the integration of Software-Defined Networking (SDN) into smart grids has emerged as a promising architectural enhancement due to its programmability, centralized control, and separation of data and control planes (Ali et al., 2020).

The dynamic nature of smart grid communication networks necessitates a recovery framework that is both resilient and adaptable to link failures. Traditional routing algorithms, which typically operate based on static topologies and pre-configured paths, are ill-suited for real-time response to failures in rapidly changing environments such as smart grids (Bhavani et al., 2023). Therefore, leveraging machine learning (ML) approaches, particularly reinforcement learning techniques such as Q-learning, has garnered attention due to their ability to model non-deterministic scenarios and improve decision-making through environmental feedback (Zhang et al., 2019). These models can dynamically update routing decisions based on observed network states, thereby minimizing the delay and packet loss typically incurred during link failures (Tang, 2024).

Among ML approaches, Q-learning is particularly suitable for routing in SDN-based smart grids due to its model-free nature and ability to optimize actions in unknown environments. Q-learning algorithms iteratively learn optimal policies by interacting with the environment, thereby enabling autonomous agents to select routes that minimize cumulative costs, including delay and energy consumption (Zhang et al., 2019). Moreover, its lightweight nature makes it implementable within the processing constraints of SDN controllers (Zhou et al., 2021). Several studies have demonstrated the efficacy of Q-learning in improving routing resilience under uncertain conditions, showing reductions in average path delay and increases in packet delivery ratio compared to traditional deterministic algorithms (Mohammadi & Javidan, 2021).

The integration of SDN and ML-based routing in smart grids is further motivated by the modular and programmable nature of SDN controllers, which allows for the deployment of intelligent decision-making layers. This architecture supports rapid failure detection and rerouting, especially when controllers are enhanced with learning modules that adapt to topological and traffic variations in real time (Abdulkadhim et al., 2022). In this context, the SDN controller is not only responsible for managing flow tables and routing paths but also acts as a host for executing learning algorithms that continuously refine network performance. Such a design is well-suited to meet the operational and security demands of critical infrastructure networks like smart grids (Mohammad, 2024).

Nonetheless, the incorporation of machine learning into network control is not without its risks. The susceptibility of ML models to adversarial attacks has raised significant concerns about the robustness and reliability of such systems (Ibitoye et al., 2019). Malicious manipulation of input data or model parameters could lead to incorrect routing decisions, service interruptions, and potential system-wide failures. Recent studies have emphasized the importance of designing secure and robust ML pipelines to mitigate adversarial threats and maintain operational integrity in mission-critical networks (Ibitoye et al., 2025). Therefore, while Q-learning and similar algorithms provide notable performance advantages, their deployment must be coupled with stringent security protocols and ongoing model validation.

From a systems perspective, SDN-enabled smart grids that incorporate ML-based routing must be evaluated on multiple performance dimensions. These include not only traditional quality of service (QoS) metrics such as end-to-end delay and packet loss, but also recovery time and algorithm execution duration following link failures (Wei et al., 2014). Studies that simulate realistic grid topologies—such as backbone structures found in German and American smart grid networks—show that Q-learning–based recovery strategies outperform conventional methods in minimizing recovery time and maintaining high service availability during failures (Mohammadi & Javidan, 2021). Furthermore, the inclusion of shared-risk link groups (SRLGs) in these simulations enables a more accurate reflection of real-world network dependencies and vulnerabilities (Ali et al., 2020).

While SDN-based failure recovery methods have been extensively studied, most existing approaches remain reactive, relying on network state polling and control-plane

computations to address faults after they occur (Ali et al., 2020). In contrast, proactive methods using reinforcement learning algorithms have demonstrated significant reductions in recovery latency by pre-computing backup paths based on predicted network states (Zhang et al., 2019). Additionally, hybrid methods that combine proactive and reactive features—such as penalizing failed links in the Q-table while simultaneously updating topology information—offer a balanced trade-off between recovery speed and computational efficiency (Mohammadi & Javidan, 2021).

In terms of implementation, platforms such as RYU, OpenDaylight, and ONOS offer different capacities for supporting intelligent routing mechanisms. While OpenDaylight and ONOS natively support proactive path installation using algorithms like Dijkstra, RYU follows a reactive model that requires flow requests to trigger path computation (Ali et al., 2020). Therefore, customizing RYU to support proactive, ML-driven routing can offer insights into enhancing controller performance and reducing flow setup latency. Moreover, integrating these systems with virtualized environments such as Mininet enables comprehensive simulation and performance benchmarking under controlled yet realistic conditions (Zheng et al., 2024).

Beyond technical performance, the adoption of intelligent, SDN-enabled routing frameworks has broad implications for the future of smart grid infrastructure. As the energy sector undergoes a digital transformation, characterized by distributed generation, electric vehicles, and IoT-enabled energy devices, the communication backbone must be equally agile and intelligent (Shafiullah et al., 2013). The use of adaptive routing techniques powered by ML offers a scalable solution to managing the increased complexity and data flows within modern grids. Moreover, the alignment of such technologies with global energy transition goals enhances sustainability and operational resilience (Bhavani et al., 2023).

In addition to operational efficiency, the integration of learning algorithms in SDN controllers also enables predictive analytics and fault prevention. For instance, deep learning models trained on historical failure patterns can predict potential future faults and reconfigure paths preemptively, minimizing service disruption (Mishra & Gupta, 2017). Furthermore, neural networks have been effectively applied in diverse fields such as radial drilling optimization and LFM signal recovery, underscoring the cross-domain versatility of these techniques (Krivoshchekov et al., 2022; Zhou et al., 2021). Their application to smart grid communication is therefore a logical and impactful extension.

However, future progress in this area depends not only on algorithmic innovation but also on hardware acceleration and system-level integration. Recent developments in in-network machine learning and hardware-based neural processing units offer promising directions for deploying low-latency, energy-efficient ML models directly within network devices (Ueyoshi et al., 2016; Zheng et al., 2024). Such developments can significantly reduce the processing burden on centralized SDN controllers and distribute intelligence across the network, thereby enhancing scalability and fault tolerance.

In conclusion, the convergence of SDN and machine learning holds transformative potential for enhancing the robustness, responsiveness, and intelligence of smart grid communication systems. Q-learning–based routing frameworks offer a compelling approach to dynamic path recovery, particularly in environments characterized by volatility and high performance requirements. However, successful implementation demands a comprehensive strategy that incorporates simulation, security, and hardware-level support. This study aims to design, implement, and evaluate a Q-learning–based routing and link failure recovery method for smart grid communication networks enabled by Software-Defined Networking (SDN).

2. Methods and Materials

2.1. Operating System

In this study, the primary objective was to utilize the Linux operating system for simulation purposes. To achieve this, the VMware virtualization tool was employed. VMware enables the execution of various operating systems, including Ubuntu 64-bit Arm version 22.04.4, virtually on a physical device. This software creates an appropriate and independent virtual environment, allowing us to conduct simulations effectively without modifying the host operating system. The allocated resources for the virtual machine are as follows:

✓ Number of processors: 8 processors, each with 2 cores

✓ Memory: 14 GB

✓ Disk space: 24 GB

2.2. Software and Programming Languages Used

For simulation purposes, Visual Studio software was used for programming. The programming of the designed unit for

the Software-Defined Network (SDN) controller was implemented using this software in the Python language and executed in the Linux terminal. AWK, a programming language for parsing and processing text files, particularly in Linux-based operating systems, was also utilized. Using AWK, large text files can be easily analyzed and subjected to the required operations in a straightforward structure. This language was used in the present study to process data and extract required information from text files. Finally, Linux shell programming was employed. The shell acts as an interface between the user and the Linux kernel and provides powerful tools to leverage kernel functionalities. Shell scripting enables users to automate repetitive operations by writing executable scripts containing commands and parameters. In network simulation projects using tools such as Mininet, shell scripts can be used to automatically launch network topologies, manage SDN controllers, and execute AWK files for data processing. These scripts automate time-consuming repetitive tasks and enhance network optimization and management.

2.3. Tools

Mininet is a network emulator designed for use in both research and development settings to create a realistic virtual network for testing before deployment on physical network hardware. Mininet is open-source, making it an ideal tool for research as it does not require a license. Unlike other simulators and emulators, Mininet is specifically designed with SDN networks in mind. It offers several features that make simulating SDN environments convenient. For instance, it allows the emulated network to be controlled by a simulated or real SDN controller. It also supports the OpenFlow protocol and can translate actual OpenFlow commands issued by an external controller through the southbound interface into simulated OpenFlow commands interpretable by emulated OpenFlow network devices.

For creating topologies, two approaches are used. The first is the command-line interface (CLI), which allows users to build network topologies in various designs (e.g., star, mesh) using specific Mininet commands and define the number of hosts and nodes directly via CLI. The second approach involves Python scripting, supported by Mininet, through which topologies can be written in Python and then executed via the CLI. Additionally, network tools like iPerf and Ping can be used to generate traffic. These tools can either be embedded in Python scripts or run via the CLI on each host node.

2.4. Proposed Method

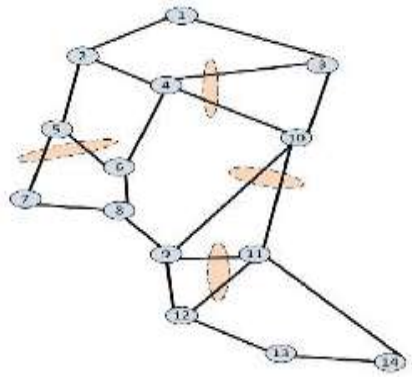
Given that maintaining communications in smart grid infrastructures is essential for delivering electricity services, a flexible communication architecture capable of recovering from link failures is required. For this reason, the present study leverages the capabilities of SDN to develop an effective method for identifying optimal sequential recovery strategies in smart grid networks. The proposed architecture comprises several components and is designed based on the SDN architecture in smart grid networks.

2.4.1. Smart Grid Network Environment

In this study, two topologies—Germany Backbone Network (GNB) and US Network (USNET)—were used. The hosts included in these topologies represent smart meters. Smart meters play a key role in smart grid networks, as they are capable of measuring, analyzing, and monitoring energy consumption. These meters continuously collect data and information regarding electricity usage and transmit it to central stations.

Figure 1

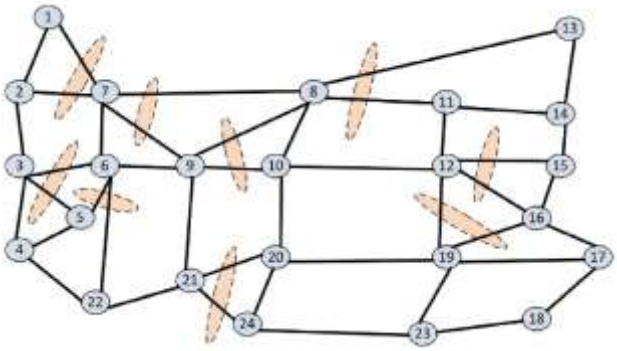
GNB Topology



In addition, they establish two-way communication with all consumers, a core feature of smart grid networks. This data exchange and bi-directional communication enable more optimized and sustainable network management. Figures 1 and 2 illustrate the topologies used in the study.

Figure 2

USNET Topology

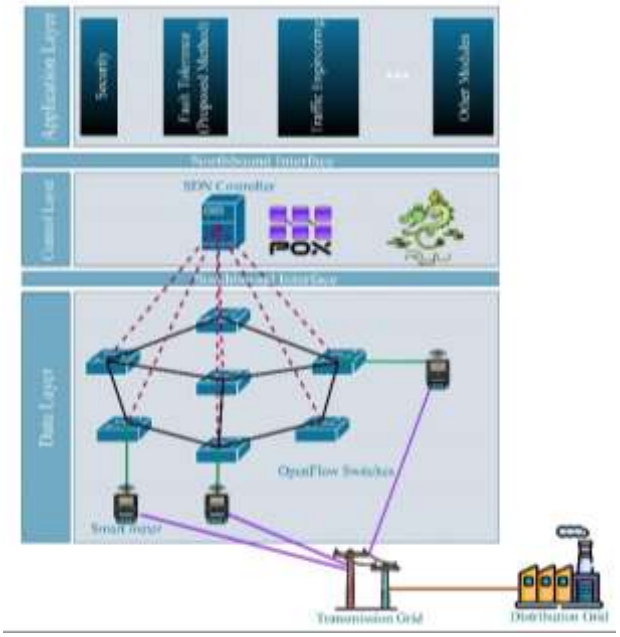


2.4.2. Role of Software-Defined Networking (SDN)

With the continuous development and widespread adoption of Software-Defined Networking (SDN), this technology also faces numerous challenges. Network failures are detrimental as they disrupt communications and significantly reduce service availability. Various factors contribute to network failures, the most common being link failure. In real-world scenarios, link disconnection is a frequent occurrence and can lead to severe consequences. When such events happen, they can cause service interruptions, negatively affect user experience, and even result in substantial financial losses. Therefore, designing a reasonable and efficient link failure recovery scheme can significantly enhance the fault tolerance, stability, and robustness of the network. Figure 3 illustrates the architecture of the proposed method based on SDN.

Figure 3

Architecture of the Proposed Method



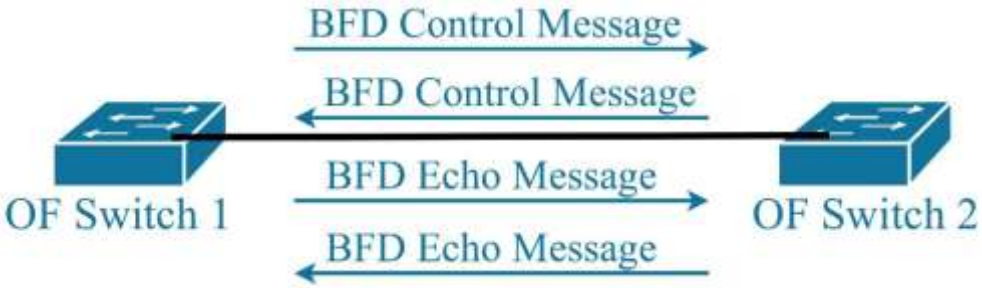
2.4.3. *Link Failure Detection Methods in Software-Defined Networks*

Generally, before initiating recovery mechanisms, link failure detection is required. In SDN, there are two common methods for detecting link failures: signal loss and bidirectional forwarding detection.

1. **Signal Loss Method**: This method is suitable for detecting failures in a specific port of the data-plane switch. When a link failure occurs, the status of the relevant OpenFlow (OF) switch port changes from "up" to "down." The associated OF switch then independently generates a port status message to inform the controller. Upon receiving this message, the controller becomes aware of the link failure and its location. In practice, this mechanism is reactive (see).
2. **Bidirectional Forwarding Detection (BFD) Method**: This method can detect both link and path failures. In this approach, two endpoint nodes periodically exchange control and echo messages. Each node responds with an echo message after receiving a control message. If a node does not receive the echo packet from the monitored connection, the link or path is assumed to have failed. Figure 4 depicts the message exchange process in the bidirectional detection method. In practice, this mechanism is proactive (see). In the present study, we use the bidirectional forwarding detection method for link failure detection.

Figure 4

Message Exchange Process in the Bidirectional Forwarding Detection Method



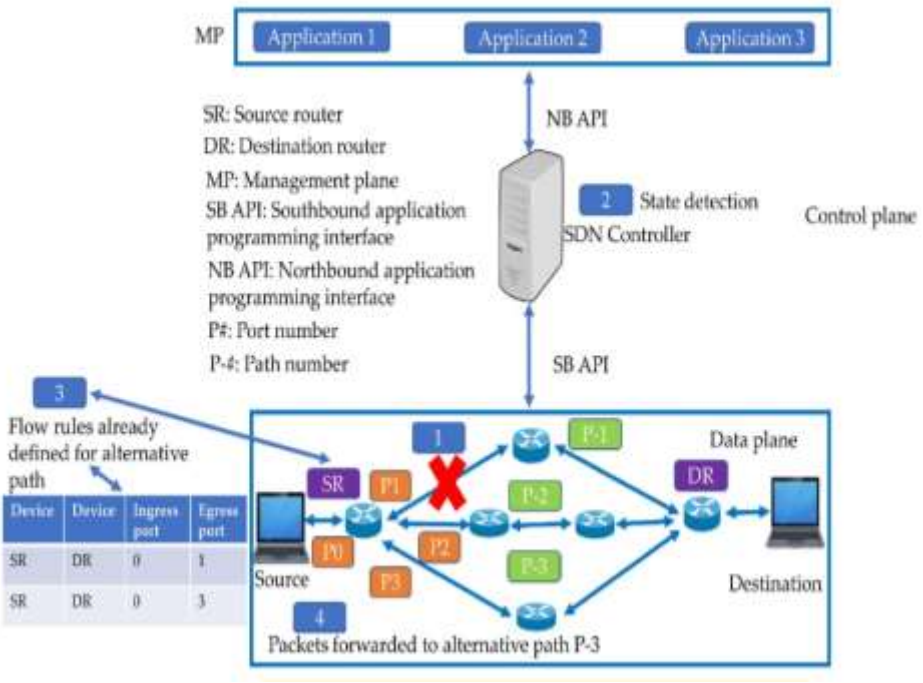
2.4.4. *Link Failure Recovery Methods in Software-Defined Networks*

When a link failure event occurs in SDN, it can be managed either proactively or reactively. During normal network operation, flows are routed from the source node to the destination node via a primary path. However, when a link fails, these two approaches handle failures differently (see).

Proactive Recovery: In proactive recovery, backup paths are pre-configured. Hence, failure detection typically occurs locally, and flows associated with the failed link are immediately redirected to the alternate path without interaction with the controller. Figure 5 illustrates link failure recovery using the proactive mechanism. When the link in path 1 fails, the flow rules for the backup path are already set on the switch; therefore, data packets are redirected from the failed link to the predefined alternate path on the switch.

Figure 5

Link Failure Recovery Using Proactive Mechanism



Proponents of proactive recovery argue that it is more efficient in terms of recovery time because the paths are pre-configured and no interaction with the control plane is necessary to find an alternative route. As a result, carrier-grade network requirements are met, meaning that paths can be restored in less than 50 milliseconds. Thus, this approach provides faster recovery without controller intervention, minimizing any delay caused by interactions with the controller to find alternate routes.

Reactive Recovery: Reactive failure recovery primarily relies on the SDN controller. Figure 6 presents a scenario of reactive link failure recovery. The following steps are taken upon detecting a link failure:

✓ The controller monitors the network status by sending periodic heartbeat messages.

✓ The controller detects any failures.

✓ The controller searches for an alternate path to replace the failed link.

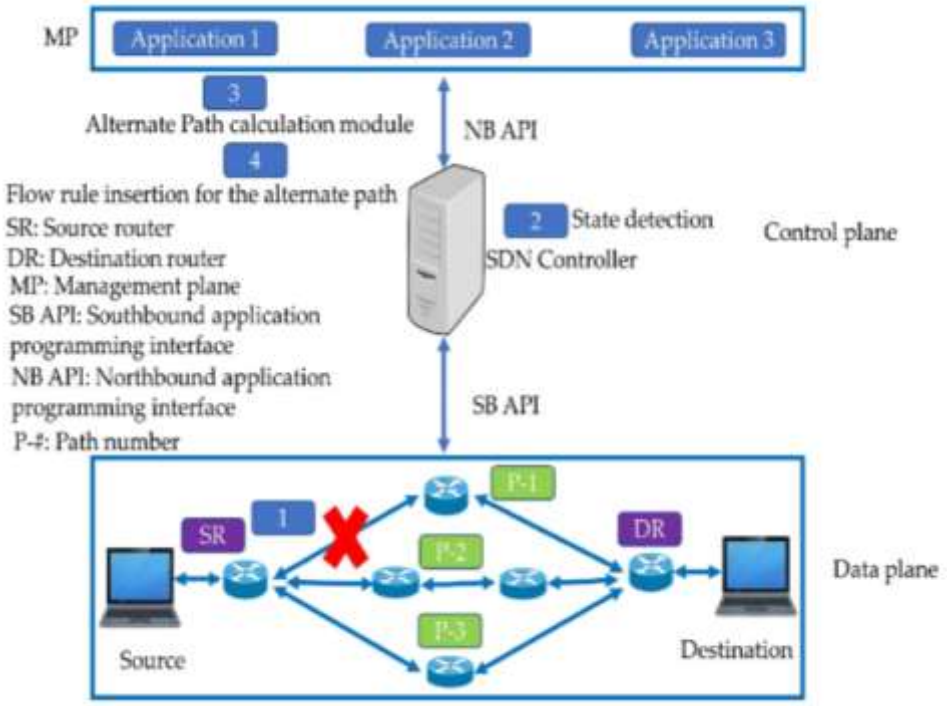
✓ The controller deletes old flow entries and installs new flow entries for the updated path in the SDN switches.

In this method, controller intervention introduces a significant delay in recovery time. This delay results from communication overhead between the switches and the controller, the additional time needed to discover an alternate path, and the time required to insert new flow entries for the updated route. Therefore, critics of this

approach argue that reactive recovery cannot meet the sub-50-millisecond delay requirements of carrier-grade networks.

Figure 6

Link Failure Recovery Using Reactive Mechanism

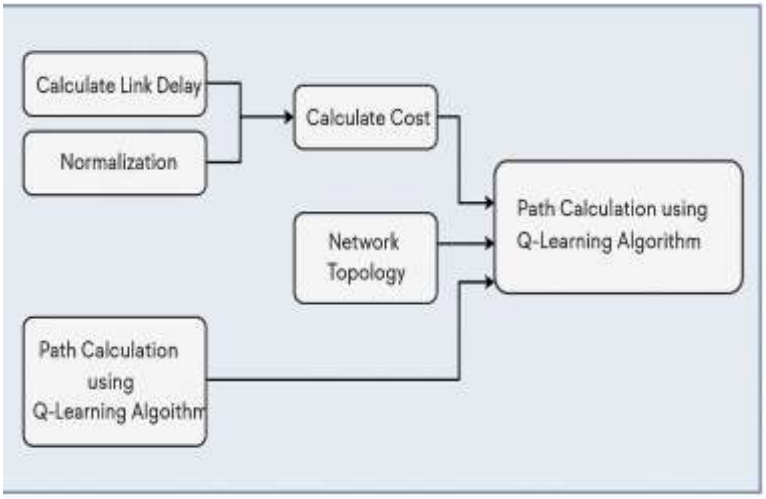


2.4.5. *Design of the Module for the Controller*

In this study, we designed a module for the controller that utilizes a recovery strategy to resolve link failures. Figure 7 illustrates the details of the fault-tolerant unit presented in Figure 3. According to Figure 7, this unit consists of seven steps.

Figure 7

Management Module Design for the Controller



✓ **Link Delay Calculation**: Information about the delay of each network link is obtained and stored in a matrix.

✓ **Link Bandwidth Calculation**: Information about the bandwidth of each network link is acquired and stored in a matrix.

✓ **Normalization**: Since delay is measured in milliseconds and bandwidth in gigabits per second, and the units are not consistent, normalization must be applied.

Equation (1)

$Cost\_s1s2 = \alpha\, d\_s1s2 + (1 - \alpha)\, b\_s1s2\text{^}{-1}$

In this formula, d represents the delay, b the bandwidth between link *s1s2*, and α is the scaling coefficient, which is set to 0.5.

✓ **Link Cost Calculation**: The cost of each link is calculated based on the normalized values of the delay and bandwidth matrices and stored in another matrix.

✓ **Network Topology**: Periodic information about the network topology—including nodes and links—is collected and stored in the relevant matrix.

✓ **Path Calculation**: The path is computed using the Q-learning algorithm.

✓ **Path Installation**: At this stage, the calculated path output is received, and the paths are installed on the switches.

The relationship between the RYU architecture and the designed module is as follows:

The designed module acts as an intelligent layer for making complex decisions about routing and enhancing network performance. This unit includes several processing functions such as calculating link delay and bandwidth, normalizing the computed values, calculating link costs, and ultimately executing a Q-learning–based routing algorithm to find the optimal path under dynamic network conditions.

In this process, the designed module depends on topology and link parameter information provided by RYU components; hence, different parts of the RYU controller play an essential role in supplying the necessary infrastructural data to this module so it can make intelligent routing decisions.

✓ **Topology Discovery**: This unit continuously discovers and updates the network topology. This information includes nodes, links, and existing paths, which are regularly provided to the designed module. In our module, this information is used as input for computing the optimal route, since the Q-learning algorithm requires awareness of the current network topology (see).

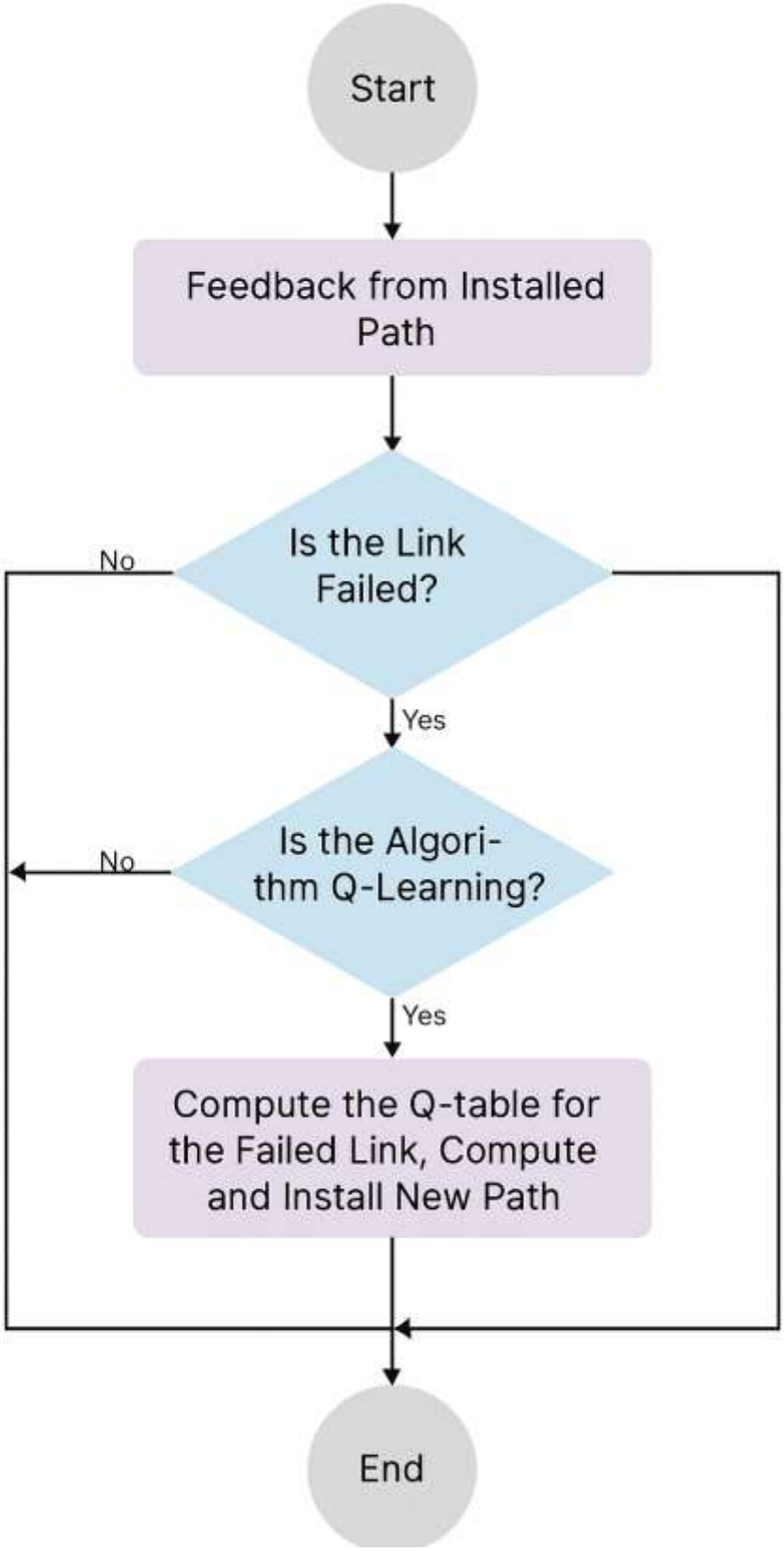
✓ **Event Management**: Upon the occurrence of any change or event in the network—such as link disconnection or reconnection—this unit sends the relevant event to other components. This capability allows the designed module to receive new parameters to recalculate paths and update optimal routing when link states change (see).

✓ **OpenFlow Parsing and Serialization**: This unit is responsible for processing OpenFlow messages and enabling communication between the controller and network devices. The module we designed utilizes this communication to install new paths, meaning that once the best path is selected by the Q-learning algorithm, the path is transmitted to the network switches through OpenFlow messages (see).

It should also be noted that upon link disconnection, feedback from the installed path is required. In the designed module, a function is implemented to detect link failure. Upon failure, a function is called to compare the new topology with the previous one, and if a change is detected, the path is recalculated and reinstalled. If Q-learning is used, a penalty value corresponding to the failure is assigned in the Q-table. Figure 8 presents the flowchart of this feedback mechanism.

Figure 8

Flowchart of the Feedback Mechanism for the Installed Path



As mentioned earlier, the Q-learning algorithm is used to improve routing and respond to link failures in the network. This algorithm leverages a Q-table, which contains value scores for different state-action pairs, to select the most optimal paths for data flow and demonstrates its capability in managing complex network conditions.

3. Findings and Results

3.1. Simulation Settings

In Figures 1 and 2, the dotted circles represent links with shared risk, meaning all links covered by a single dotted circle are exposed to a similar risk. In the GNB and USNET topologies, we considered four and nine groups of shared-risk links, respectively. In both experiments, we implemented a Python script in Linux to simulate link failures. This script uses the ifdown command to disconnect a link and the ifup command to reconnect it. The script is activated every 10 seconds and randomly selects one shared-risk link from each group, disconnecting it for a specific period defined as failure time. After the failure time, the script reconnects the previously disconnected links. In all simulations of the proposed method, the controller is configured to poll network statistics, execute the proposed method every 20 seconds, and announce separate routes for each source-destination pair to the relevant switches. It is noteworthy that the simulation duration is set to 240 seconds (4 minutes).

In the GNB topology, the delay and bandwidth of each link are configured to 10 milliseconds and 5 megabits per second, respectively. This topology includes three source-destination pairs, with each source host generating UDP traffic at a rate of 1 megabit per second (with a packet size of 1000 bytes) and sending it to the destination host. In the USNET topology, each link's delay and bandwidth are configured to 20 milliseconds and 10 megabits per second, respectively. This topology has five source-destination pairs, and each source host generates UDP traffic at a rate of 2 megabits per second (with a packet size of 1000 bytes) and sends it to the destination. All simulation configurations are aligned to enable comparison with the reference article.

3.2. Computational Parameters

The simulation results are analyzed based on quality of service parameters (average end-to-end delay and packet loss), recovery time, and algorithm execution time. These parameters are defined as follows:

✓ *Average end-to-end delay*: Refers to the average time required for successful transmission of each data packet from the source to the destination.

✓ *Packet loss*: Indicates the ratio of total data packets that did not reach the destination to the total number of packets sent from the traffic source.

✓ *Recovery time*: Represents the duration required to detect a link failure and compute a new path.

✓ *Algorithm execution time*: In the controller module, the start and end times of the path computation are recorded, and the difference is used to determine the algorithm's execution time.

3.3. Evaluation of the Proposed Method Versus the Reference Article

This section presents the evaluation of the results for the GNB and USNET scenarios and compares them with the reference method.

3.3.1. Evaluation of Link Failure Results in GNB and USNET Topologies

In Figures 9 and 10, the results of the reference article clearly demonstrate superior performance in reducing average end-to-end delay through the use of optimization techniques. These techniques include improvements in routing and the selection of shorter and more efficient paths for data flows in the network, which result in reduced packet transmission time from source to destination and lower overall network delay. This reduction in delay directly enhances network performance under various conditions and demonstrates the reference method's superiority in minimizing delay.

In contrast, the proposed method in this study employs the Q-learning algorithm for recovering failed links. In this algorithm, whenever a link fails, a penalty is assigned to the corresponding Q-table entry. These penalties prompt the system to automatically choose alternate paths to avoid failed links. If the available alternate links in the network are longer or contain more nodes, the selected path will also be longer. Increased path length directly leads to higher network delay, as packets must traverse more nodes.

Figure 9

Average End-to-End Delay for the GNB Topology

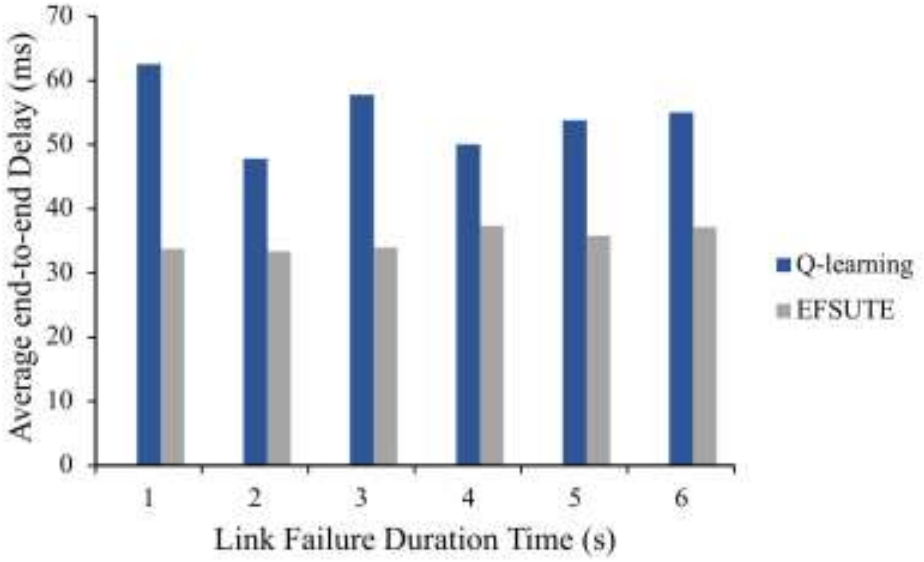
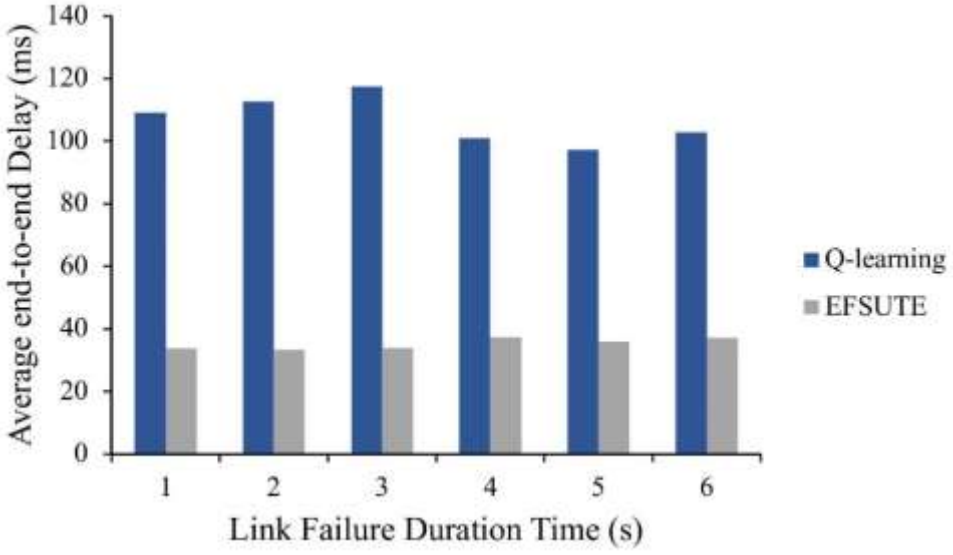


Figure 10

Average End-to-End Delay for the USNET Topology



In Figures 11 and 12, it is observed that the packet loss rate in the proposed method has decreased on average by 66.62% in the German topology and 90.50% in the U.S. topology compared to the reference article. This performance improvement is attributed to the higher algorithm execution time in the reference article; the process of detecting a link failure and computing and installing a new path takes significantly longer. This delay results in the loss of a large number of packets across the network, thereby negatively affecting system performance.

Figure 11

Packet Loss for the GNB Topology

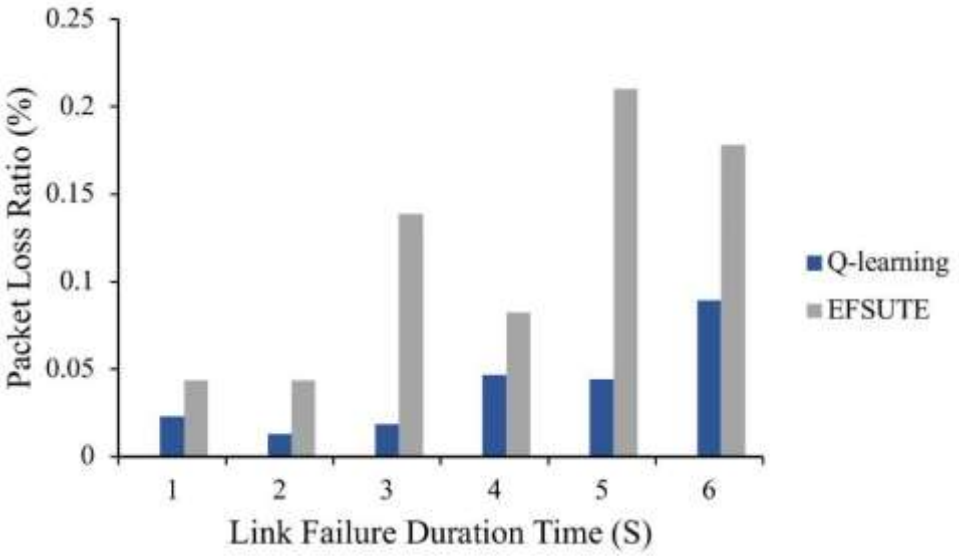
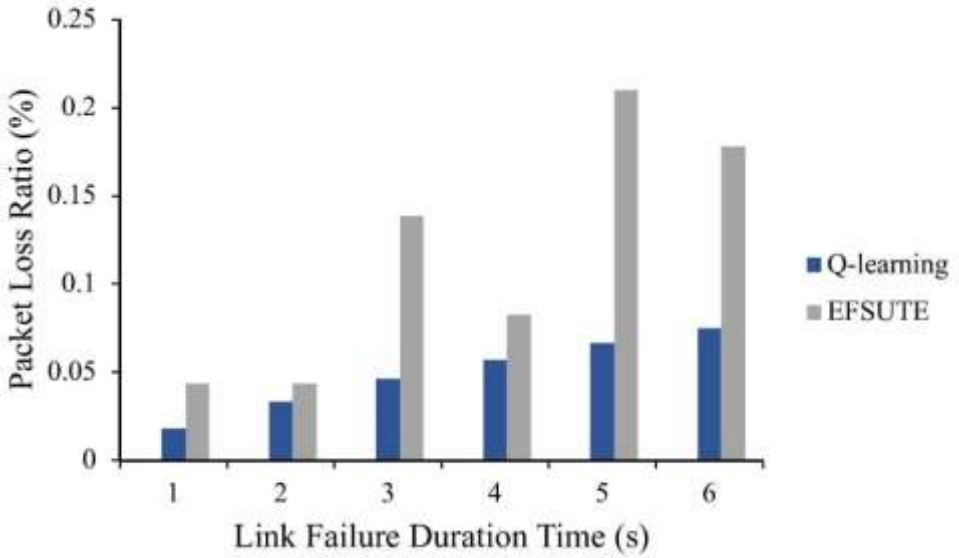


Figure 12

Packet Loss for the USNET Topology



Recovery time is defined as the sum of the path installation time and algorithm execution duration. In the reference article, the algorithm's execution time is high due to the complexity of the failure detection process and the computation of new paths, which can significantly extend the recovery time. Conversely, in the proposed method, the use of the Q-learning algorithm and improved processes for detection and path selection have effectively reduced the algorithm's execution time. As shown clearly in Figures 13 and 14, the average recovery time in the proposed method has been reduced by 85.99% in the German topology and 76.99% in the U.S. topology compared to the reference article.

It is important to note that to better compare the results of the two methods, due to the significant difference in recovery time scales, two separate y-axes are used in the graphs: the left axis for values from the Q-learning algorithm and the right axis for values from the reference method (EFSUTE). This configuration enables more accurate observation and analysis of the results.

Figure 13

Recovery Time for the GNB Topology

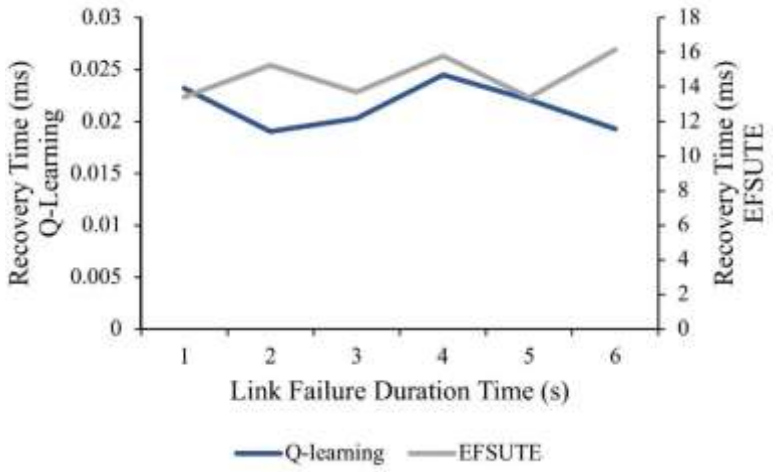
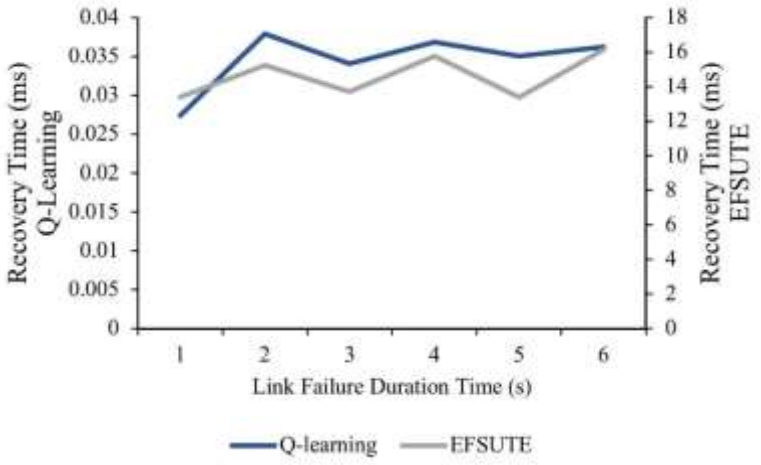


Figure 14

Recovery Time for the USNET Topology



The algorithm execution time in the proposed method has significantly decreased compared to the reference article. As shown in Figures 15 and 16, the execution time was reduced by 91.99% in the German topology and 98.99% in the U.S. topology during the experiments. This substantial difference is due to the time complexity of the algorithm in the reference article, which is classified as non-deterministic polynomial (NP) and thus requires significantly more runtime than the Q-learning algorithm. In contrast, the proposed method benefits from the Q-learning algorithm's lower time complexity and has demonstrated better performance in terms of execution speed.

It is again worth noting that for more effective comparison, due to the substantial difference in recovery time scale, the graphs utilize two separate y-axes: the left for the Q-learning algorithm values and the right for the reference article (EFSUTE). This setting facilitates more precise visualization and interpretation of results.

Figure 15

Algorithm Execution Time for the GNB Topology

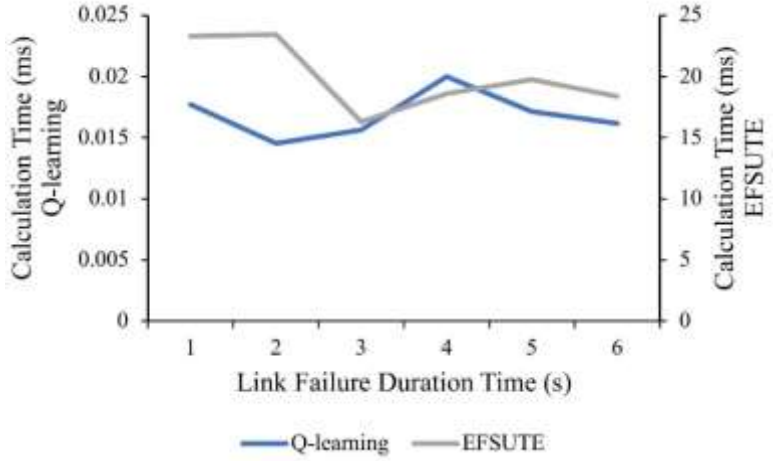
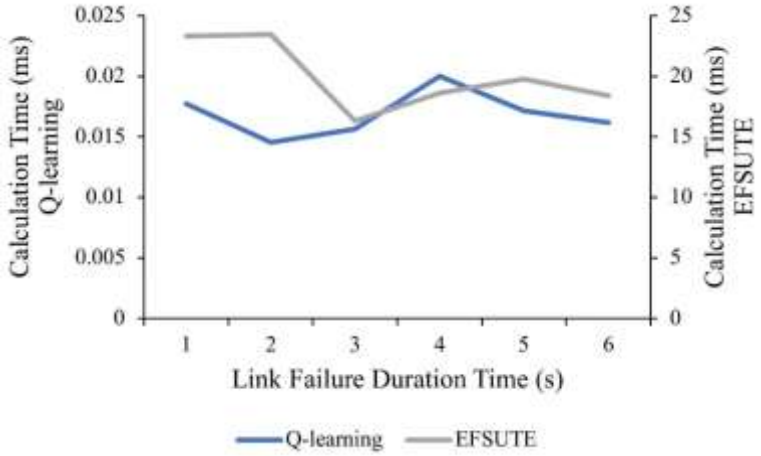


Figure 16

Algorithm Execution Time for the USNET Topology



3.3.2. Evaluation of Two-Link Failure Results in GNB and USNET Topologies

As observed in Figures 17 and 18, the average end-to-end delay in the normal method is lower than in the proposed method, because the routing algorithm in the normal method operates based on the number of hops and always selects the shortest path. Since the link delay values are identical in both methods, selecting the shortest path inherently results in the lowest delay. In contrast, in the proposed method, due to the penalization of the Q-table associated with the failed link, the path to the destination becomes longer, thereby increasing overall network delay.

Figure 17

Average End-to-End Delay for the GNB Topology

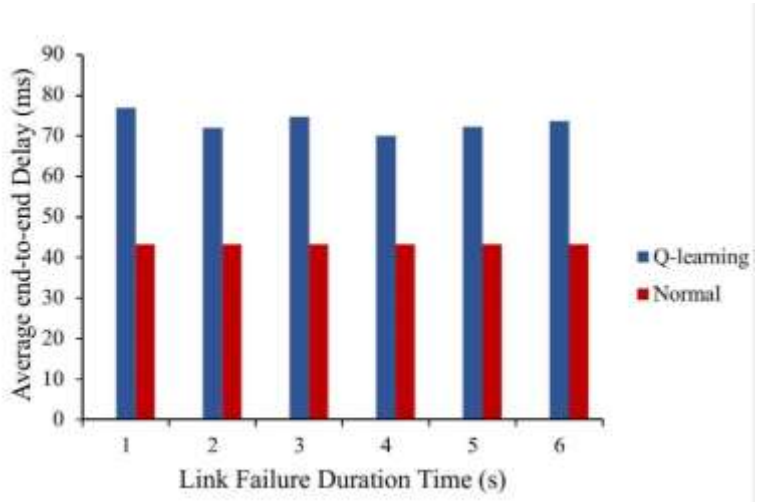
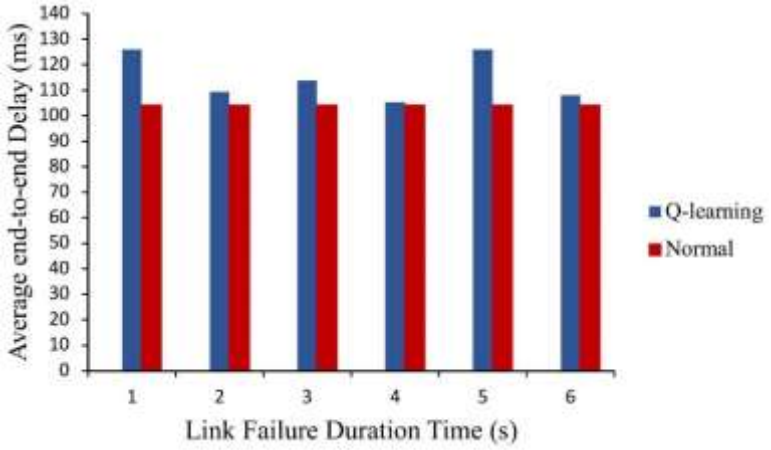


Figure 18

Average End-to-End Delay for the USNET Topology



As shown in Figures 19 and 20, the packet loss rate in the proposed method is reduced on average by 67.95% in the German topology and 13.15% in the U.S. topology compared to the normal method. In the normal method, routing is performed solely based on the shortest path, and when a link failure occurs, no new alternative path is installed. This leads to the loss of a significant number of packets. However, in the proposed method, upon detecting a link failure, the Q-table entry related to the failed link is penalized to prevent its reuse, and instead, a new path is selected from among the healthy links. This process effectively reduces packet loss.

Figure 19

Packet Loss for the GNB Topology

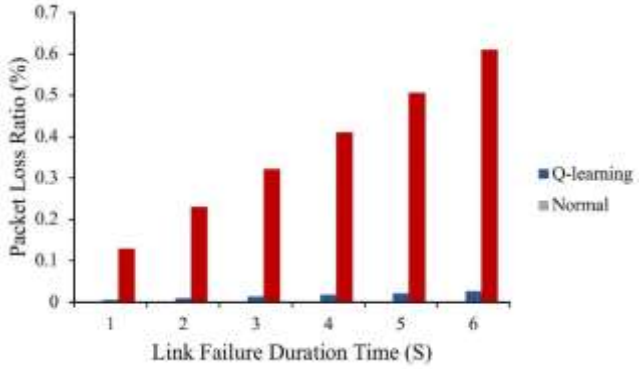
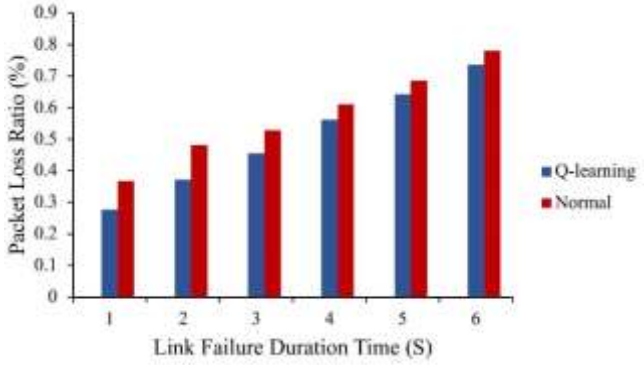


Figure 20

Packet Loss for the USNET Topology



As stated earlier, in smart grids, the priority is to ensure that data is accurately received and not lost. Therefore, the primary focus of this study is on reducing the packet loss rate, since this phenomenon can have severely detrimental effects.

3.4. Normal Method

In many SDN controllers, including OpenDaylight and ONOS, a proactive routing mechanism is implemented. These controllers, using algorithms such as Dijkstra, compute optimal paths for data flows in advance and install them on the switches. This allows data packets to be transmitted quickly, without the need for additional processing for each new request. As a result, network performance is significantly enhanced and transmission delays are reduced.

However, the RYU controller used in this study does not have such a built-in proactive routing mechanism. RYU operates using a reactive routing approach, meaning that when a new data flow arrives at a switch without a predefined path, the switch sends a request to the controller. At this point, the controller computes the required path and communicates it to the switches so that the packets can be forwarded accordingly. This process is repeated separately for each new flow. This reactive approach increases the processing overhead on the controller, as RYU must compute paths step-by-step for every new flow.

To overcome this limitation and align the performance of the RYU controller with that of other controllers, routing using the Dijkstra algorithm—referred to as the "normal method"—has been implemented in this study. This measure helps improve the performance and efficiency of the RYU controller. Furthermore, in the comparison between the proposed method and the reference article, only one link from a shared-risk group was disconnected. In contrast, in this comparison with the normal method, both links in the shared-risk group are disconnected, and the resulting outcomes are discussed in the next section. It is important to note that the normal method was developed as an auxiliary,

external part of the main study and is intended to highlight the reduction in packet loss rate.

4. Discussion and Conclusion

This study aimed to evaluate the performance of a Q-learning–based routing strategy for smart grid networks enabled by Software-Defined Networking (SDN). The simulation results in both GNB and USNET topologies confirmed that the proposed method, despite a trade-off in end-to-end delay, significantly outperforms traditional and reference approaches in packet loss reduction, recovery time, and algorithm execution time.

First, regarding end-to-end delay, the results show that the normal method, which relies on shortest-path routing based on hop count, achieved lower average delays compared to the proposed method. This outcome is expected because, in the normal method, routing is static and always selects the path with the fewest hops, inherently minimizing delay when link qualities are equal. In contrast, the Q-learning–based method imposes penalties on failed links within the Q-table, which can result in the selection of longer alternate paths to avoid failed segments. Consequently, as more nodes are traversed, delay increases. However, this is a calculated trade-off in favor of overall resilience and packet delivery success. Such trade-offs have been acknowledged in earlier studies on intelligent routing in SDN networks, where the primary goal was maximizing reliability and delivery rates under dynamic network conditions rather than minimizing delay alone (Ali et al., 2020; Zhang et al., 2019).

Second, the proposed method demonstrated a significant reduction in packet loss, particularly under multiple-link failure scenarios. In the GNB topology, the average packet loss decreased by approximately 67.95%, and in the USNET topology, by 13.15% when compared with the normal method. This result underscores the primary strength of Q-learning in dynamic environments. Unlike traditional routing algorithms that do not adapt to network state changes during runtime, reinforcement learning methods like Q-learning are capable of reacting to disruptions by learning from past failures and avoiding problematic links. These findings are consistent with previous research, which highlighted the capacity of Q-learning to improve network robustness in failure-prone environments, especially in smart grids where real-time data delivery is mission-critical (Bhavani et al., 2023; Zhang et al., 2019). Moreover, machine learning–based approaches have proven effective in failure detection and adaptive routing across various domains, as shown in both power distribution modeling (Wei et al., 2014) and radar network recovery (Zhou et al., 2021).

Third, the study's most pronounced performance gain was in the area of recovery time and algorithm execution time. In comparison with the EFSUTE algorithm from the reference article, the proposed Q-learning method achieved a recovery time reduction of 85.99% in the GNB topology and 76.99% in the USNET topology. Similarly, algorithm execution time was reduced by 91.99% and 98.99%, respectively. These improvements are particularly noteworthy considering the operational importance of rapid failure recovery in smart grid environments. Delays in rerouting after link failures can severely impact grid responsiveness, leading to cascading system errors or energy delivery mismatches. Q-learning's low computational complexity and its model-free nature allow for rapid decision-making and policy updates, which are vital in such time-sensitive scenarios. Previous works also support this assertion, stating that Q-learning provides faster adaptability in real-time applications compared to deterministic or computationally heavy multi-objective routing strategies (Mishra & Gupta, 2017; Mohammadi & Javidan, 2021).

Another important observation was the effectiveness of penalization within the Q-table during link failure events. This mechanism enables the controller to adaptively avoid previously failed links and reroute traffic through alternate paths, even if those paths are longer. The reactive-reflexive behavior of the Q-agent in the SDN controller reinforces fault avoidance and demonstrates a form of experiential learning not available in traditional or statically preconfigured routing approaches. The value of such adaptive strategies in critical systems has been emphasized by several studies, particularly those addressing the design of resilient and secure SDN architectures (Abdulkadhim et al., 2022; Mohammad, 2024).

Furthermore, the integration of SDN with Q-learning aligns with the broader shift toward intelligent, decentralized network control, allowing networks to self-optimize in response to evolving topologies and threats. This convergence is especially relevant in smart grids, where the control infrastructure must manage a diverse array of nodes—from smart meters to substations—and ensure fault-tolerant data transmission. Reinforcement learning strategies like the one proposed here fulfill this requirement by incorporating environmental feedback into future decision-

making, reducing network overhead, and enhancing scalability (Ibitoye et al., 2025; Tang, 2024).

While the results of the proposed method are promising, it is important to consider the implications of security in machine learning–driven SDN controllers. Recent research has highlighted the vulnerability of such models to adversarial attacks that can manipulate input data or training sets to compromise routing decisions (Ibitoye et al., 2019). Although Q-learning is generally less complex and thus potentially less vulnerable than deep learning models, it is still susceptible to reward manipulation and spoofed feedback. As such, embedding robust adversarial detection and countermeasures is crucial for operational deployment, particularly in safety-critical domains like energy infrastructure (Ibitoye et al., 2025; Ueyoshi et al., 2016).

The simulation environment also played a critical role in validating the results. By utilizing Mininet and simulating shared-risk link groups (SRLGs) within realistic topologies such as GNB and USNET, the study approximated real-world network vulnerabilities more accurately than models relying on simplified topologies. The use of heartbeat messaging for link monitoring, and the regular update of the SDN controller's Q-table, facilitated a robust test of the proposed architecture's responsiveness and reliability. This approach is consistent with best practices in ML-in-network experimentation, as suggested by recent studies on in-network ML inference platforms like Planter (Zheng et al., 2024).

In conclusion, the study demonstrates that Q-learning–based routing in SDN-enabled smart grids can significantly enhance resilience, especially in failure-prone scenarios. While it introduces modest increases in delay due to longer alternate paths, this is outweighed by substantial improvements in packet loss reduction, recovery time, and computation efficiency. These findings affirm the relevance of reinforcement learning for self-adaptive, fault-tolerant networking, providing a clear path forward for integrating ML into smart grid control infrastructures (Krivoshchekov et al., 2022; Mohammad, 2024; Mohammadi & Javidan, 2021).

Despite its promising outcomes, the present study has several limitations. First, the simulations were conducted in a virtualized testbed environment, which may not capture all real-world complexities such as hardware-induced latencies, varying traffic intensities, or physical environmental interferences. Second, the learning model assumes full observability and timely reception of feedback, which may not always be feasible in real-time smart grid scenarios. Third, only Q-learning was evaluated; the inclusion of other reinforcement learning algorithms such as Deep Q-Networks (DQN) or Actor-Critic models may offer additional insights. Moreover, the scope of adversarial risks and security vulnerabilities was not empirically tested in this implementation and requires further exploration.

Future research should aim to expand the study by incorporating hybrid learning approaches, such as combining Q-learning with predictive analytics based on historical fault data. Additionally, exploring the integration of hardware-accelerated ML platforms can help address latency issues, particularly in edge computing settings. Long-term studies under real-time conditions, including stress testing under simultaneous multi-node failures or cyberattacks, will also provide a more comprehensive understanding of the model's resilience. Comparative studies between different SDN controllers and their capability to host ML-based decision modules will further clarify deployment feasibility.

Practitioners aiming to deploy intelligent routing in SDN-enabled smart grids should prioritize modular design, allowing easy integration of Q-learning models with existing controllers. They should also implement periodic topology monitoring and update mechanisms to ensure Q-tables remain relevant. Proactive route learning, especially under shared-risk link scenarios, can reduce failure response times and enhance service availability. Finally, adopting a security-first approach, including model validation and feedback authentication, will be critical to ensuring trust and performance in real-world deployments.

Authors' Contributions

Authors contributed equally to this article.

Declaration

In order to correct and improve the academic writing of our paper, we have used the language model ChatGPT.

Transparency Statement

Data are available for research purposes upon reasonable request to the corresponding author.

Acknowledgments

We would like to express our gratitude to all individuals helped us to do the project.

Declaration of Interest

The authors report no conflict of interest.

Funding

According to the authors, this article has no financial support.

Ethics Considerations

In this research, ethical standards including obtaining informed consent, ensuring privacy and confidentiality were considered.

References

Abdulkadhim, M., Abdulmuhsen, N. Q., & Al-Kadhimi, A. M. (2022). Design and Simulation of a Software Defined Networking-Enabled Smart Switch for Internet of Things-Based Smart Grid. *Indonesian Journal of Electrical Engineering and Computer Science*, *25*(2), 780. https://doi.org/10.11591/ijeecs.v25.i2.pp780-787

Ali, J., Lee, G. M., Roh, B. H., Ryu, D. K., & Park, G. (2020). Software-defined networking approaches for link failure recovery: A survey. *Sustainability*, *12*(10), 42-55. https://doi.org/10.3390/su12104255

Bhavani, A., Ramana, A. V., & Chakravarthy, A. S. N. (2023). A Review on Machine Learning Based Routing Protocols for Delay Tolerant Networks. *Intelligent Decision Technologies*, *17*(2), 287-299. https://doi.org/10.3233/idt-220018

Ibitoye, O., Abou-Khamis, R., Shehaby, M. e., Matrawy, A., & Shafiq, M. O. (2019). The Threat of Adversarial Attacks on Machine Learning in Network Security -- A Survey. https://doi.org/10.48550/arxiv.1911.02621

Ibitoye, O., Abou-Khamis, R., Shehaby, M. e., Matrawy, A., & Shafiq, M. O. (2025). The Threat of Adversarial Attacks Against Machine Learning in Network Security: A Survey. *J. Electron. Electric. Eng.* https://doi.org/10.37256/jeee.4120255738

Krivoshchekov, S., Kochnev, A., & Ozhgibesov, E. (2022). The Application of Neural Networks to Forecast Radial Jet Drilling Effectiveness. *Energies*, *15*(5), 1917. https://doi.org/10.3390/en15051917

Mishra, C., & Gupta, D. (2017). Deep Machine Learning and Neural Networks: An Overview. *Iaes International Journal of Artificial Intelligence (Ij-Ai)*, *6*(2), 66. https://doi.org/10.11591/ijai.v6.i2.pp66-73

Mohammad, R. (2024). *Principles of designing software-defined networks*. Abou Ali Sina University Press.

Mohammadi, R., & Javidan, R. (2021). EFSUTE: A novel efficient and survivable traffic engineering for software defined networks. *Journal of Reliable Intelligent Environments*, *137*, 1-14. https://link.springer.com/article/10.1007/s40860-021-00139-0

Shafiullah, G. M., Maung Than Oo, A., Ali, A. B. M. S., & Wolfs, P. (2013). Smart grid for a sustainable future. *Smart Grid and Renewable Energy*, *4*, 23-34. https://doi.org/10.4236/sgre.2013.41004

Tang, Y. (2024). The Application of Machine Learning in the Field of Network Security. *Tcsisr*, *7*, 363-369. https://doi.org/10.62051/cw085k64

Ueyoshi, K., Marukame, T., Asai, T., Motomura, M., & Schmid, A. (2016). Robustness of Hardware-Oriented Restricted Boltzmann Machines in Deep Belief Networks for Reliable Processing. *Nonlinear Theory and Its Applications Ieice*, *7*(3), 395-406. https://doi.org/10.1587/nolta.7.395

Wei, Y., Ji, C., Galvan, F., Couvillon, S., Orellana, G., & Momoh, J. A. (2014). Learning Geotemporal Nonstationary Failure and Recovery of Power Distribution. *IEEE Transactions on Neural Networks and Learning Systems*, *25*(1), 229-240. https://doi.org/10.1109/tnnls.2013.2271853

Zhang, Y., Wu, J., Chen, Z., Huang, Y., & Zheng, Z. (2019). Sequential node/link recovery strategy of power grids based on q-learning approach. In 2019 IEEE International Symposium on Circuits and Systems (ISCAS),

Zheng, C., Zang, M., Hong, X., Perreault, L. P. L., Bensoussane, R., Vargaftik, S., Ben-Itzhak, Y., & Zilberman, N. (2024). Planter: Rapid Prototyping of in-Network Machine Learning Inference. *Acm Sigcomm Computer Communication Review*, *54*(1), 2-21. https://doi.org/10.1145/3687230.3687232

Zhou, Y., Zhang, F., Sun, G., & Pan, S. (2021). Machine Learning Based LFM Signal Recovery for Fiber-Connected Radar Networks. M4B.2. https://doi.org/10.1364/oecc.2021.m4b.2